Generating Guitar Tablature with LHF Notation via DGA and ANN

Daniel R. Tuohy and W.D. Potter

University of Georgia, Athens GA 30605, USA,
`daniel2e@uga.edu`

Abstract. This paper describes a system for converting music to guitar tablature. At run time, the system employs a distributed genetic algorithm (DGA) to create tablature and an artificial neural network to assign fingers to each note. Three additional genetic algorithms are used to optimize the fitness function of the DGA, the operating parameters of the DGA, and the learning environment of the Neural Network. These steps are taken in the hope of maximizing the consistency of our algorithm with human experts. The results have been encouraging.

1 Introduction: The Guitar Fingering Problem

Stringed instruments suffer from a one-to-many relationship between notes and the positions from which they can be played. Figure 1 shows four different positions from which the same "E" can be produced. These are called fretboard positions and are described by two variables, a string and a fret. It is the burden of the performer to choose a fretboard position for every note in a piece of music. Tablature is a musical notation system that represents music as fretboard positions rather than as notes, freeing the performer from tedious decision making. Tablature is ordinarily created by humans. There have been commercial attempts to generate tablature automatically, but these are notorious for creating unnecessarily difficult or unplayable tablature [3] [11].
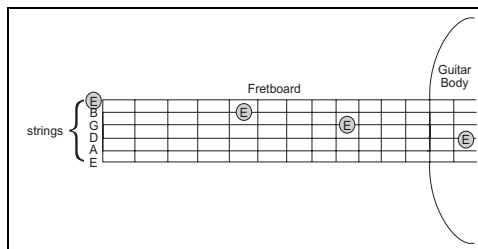

Fig. 1. Four different fretboard positions for a 330 hz "E"

It is the goal of our system to be able to consistently generate tablature competitive with that created by humans. This problem has been given considerable

attention in the last few years. Examples of three recent academic attempts to achieve a similar goal were conducted at the Universities of Doshisha, Victoria, and Torino, each with apparent weaknesses. The model from the University of Doshisha has been designed only for use on monophonic melodies, and cannot create tablature for pieces including chords [3]. The model from the University of Victoria is a learning algorithm that consults published tablature to learn different musical styles, and an assessment of performance on an independent test set is not provided [6]. We will only compare our results to those from the University of Torino, as this approach has the same aspirations as our own [5][4].

In this paper we will describe four genetic algorithms and one neural network. In section two we will describe TabGA, the distributed genetic algorithm designed to create tablature from guitar music. Section 2 also provides an analysis of PMGA, a Meta GA for optimizing the operating parameters of TabGA. A Meta GA, as used in this paper, is defined as a GA that executes another GA in its fitness evaluations. Section three is an analysis of the fitness function of TabGA, upon which the ability to create tablature competitive with human experts is entirely dependent. In section four we describe FFMGA, a Meta GA used to tune the fitness function of TabGA. In section five we discuss LHFNet, the neural network we have implemented to determine appropriate left handed fingering for guitar tablature. Section 5 also will address NNGA, a GA used to determine the architecture for LHFNet. An overview of the entire system is illustrated in Figure 2.
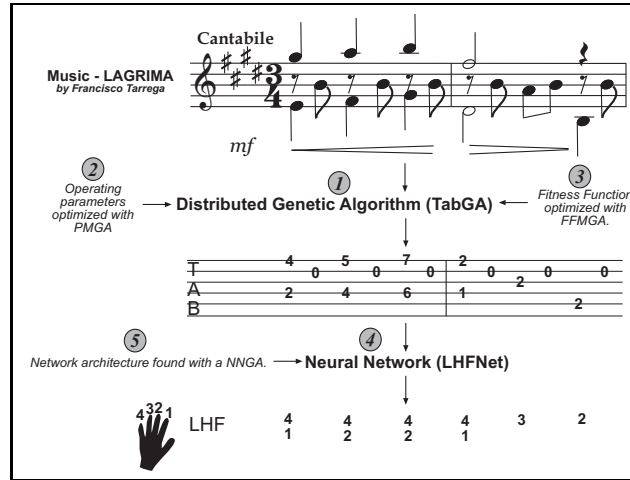

Fig. 2. An overview of our system and it's five components. (1) TabGA takes as input music and creates tablature. TabGA's operating parameters were set using (2) PMGA and the fitness function was optimized with (3) FFMGA. The resulting tablature is then used by (4) LHFNet to create left handed fingering notation. The architecture of LHFNet was discovered with (5) NNGA.

2 TabGA: Finding Good Tablature with Distributed Genetic Search

2.1 A GA for the Tablature Problem.

In previous work we detailed how genetic algorithms can be used to find good tablature [10]. At the start of a run, a seed population of hundreds of random, but musically valid, tablatures is created. The workings of genetic algorithms are well-known, and figure 3 illustrates how crossover and mutation function in this domain.

2.2 Why we use Genetic Search.

Suppose that, on average, a note can be played in three different fretboard positions (this is lower than the actual average). Then a tablature with n notes has 3^n possible tablatures. Exhaustive search techniques are quickly rendered impractical as the number of notes is increased. One might assert that this does not present a problem, since there does not seem to be a reason to create tablature for more than a few notes at a time. Why should we be concerned with notes to be played several seconds in the future when deciding on a fretboard position for a given note? Consider the 1st and 10th notes in a piece of music. The fretboard position of the 1st note is only directly dependent on those of the few notes that immediately follow it, say only as far as the 4th note. However, the 4th note may depend upon the 7th, and that on the 10th and so on until there is a break in the music that allows the performer time to move his hand freely. This establishes a dependency chain within phrases that makes it inadvisable to assign positions to notes sequentially, as is the approach (and weakness) of many commercial tablature generators. For this reason, we choose to break music into logical phrases manually, then process each phrase with the GA.

2.3 Genitor II Background

The DGA that we have implemented in TabGA is based on the Genitor II system devised by Whitley and Starkweather [14]. Genitor II is an implementation of a DGA, also called Island GAs. It was chosen because of its reliability for converging quickly on very good solutions over a wide variety of NP-hard problems [1]. Adherence to the steady-state and island paradigms is characteristic of Genitor II. A Steady State GA is one in which individuals are introduced into the population individually, rather than as a whole new generation. An Island GA is one in which the population is divided into separate sub-populations, each of which conducts genetic search independently. It is this aspect which provides the distributed nature of the DGA. Each island consists of a small population of individuals and uses a Steady State GA to evolve that population. Because of sampling bias inherent in the initial population of each island, the islands will naturally focus on different parts of the search space. The islands periodically swap their best individuals between one other, which encourages the sharing
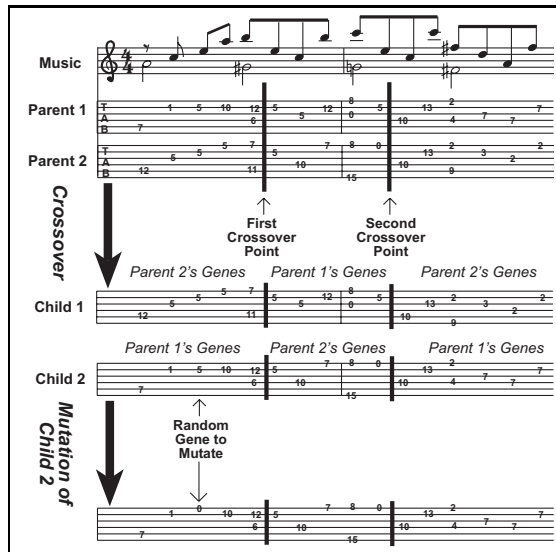
Fig. 3. Obtaining a child from two parents in a simple GA.

of beneficial characteristics that might proliferate in distant parts of the search space.

Whitley specifically mentioned that problems with "multiple structurally incompatible solutions" are well-suited for exploitation by the Genitor II scheme. The application he has in mind is the optimization of weights in a neural network. Such problems may have many globally competitive solutions that, when recombined with each other, yield very poor results. This is exactly the problem that we face with tablature generation. Consider a piece of music that is optimally played low on the fretboard, but could be as easily played higher on the fretboard. Attempting to recombine a tablature of each type would result in an unplayable tablature. The results presented by Whitley and Starkweather on neural network optimization indicate that Genitor II is better suited to such problems than other genetic search techniques. The Genitor II scheme presents us with more operating parameters than would a simpler evolutionary search technique, so we again turn to genetic search to find good values for these parameters.

2.4 Optimization with the Parameter Meta GA (PMGA)

A set of DGA operating parameters can be represented as a chromosome composed of eight variables. These are the population size, the number of islands, the interval at which islands swap individuals, the number of individuals that are swapped, the linear bias of the ranked-based selection, two variables governing adaptive mutation, and a binary variable indicating whether to use one or two point crossover. In PMGA we have chosen to use a meta GA based loosely

on Whitley's Genitor I (non-distributed) approach to search for the best set of values for TabGA [13].

To assess fitness, a DGA is run on eleven different musical excerpts. For each excerpt, we evolved a population of tablatures until no improvement was found in the best individual for 5000 replacements. We previously ran a hand tuned DGA 100 times on all eleven excerpts to establish a baseline tablature for each excerpt that should be close to an optimal solution as defined by our fitness function. We compare the best individual at the end of the evolution process with the baseline.

PMGA is a steady-state scheme with ranked based selection and a population size of 150. Uniform crossover and average (arithmetic) crossover are used interchangeably since both are applicable. Mutation is random and occurs with a probability of 10% at each gene. We ran PMGA for 20,000 replacements, at which point very little improvement in average population fitness was being made. We then ran each of the 150 DGA parameter sets in the final population ten times and averaged their performance, taking the DGA with the best average performance as our final solution.

3 An overview of the TabGA Fitness Function for evaluating tablature difficulty

The fitness function for our GA is required to assess the difficulty, or "playability", of tablature. We have elected not to use a system based on minimizing the distance traveled for each finger. This method was first detailed by Sayegh and has been expounded upon in other research [7] [6] [5]. Systems that adopt this paradigm analyze of the movement of each finger, and then use a cost function that penalizes each finger's movement. However, net finger movement alone does not constitute difficulty. Many other factors, some very difficult to articulate, are also important. We have implemented a fitness function based on the author's musical expertise. We have endeavoured to include in our fitness function as many relevant difficulty factors as possible, though none are dependent upon accurately accounting for each finger.

We will now provide an overview of our fitness function, which is quite lengthy. We will provide as much information as possible, but will summarize specific details for the sake of brevity. We have included as many metrics of difficulty as possible, and will allow a Meta GA (FFMGA) to decide the importance that each metric should be given.

3.1 Biomechanical Factors

There are twelve factors considered to be biomechanical in nature because they assess the physical difficulty of hand movement.

How the Hand Moves. The first eight biomechanical factors are concerned with the total amount of movement the hand must undertake. These are the

number of times frets must be depressed, a reward for the number of times open strings are played, the total fretwise distance between sequential chords, a penalty for larger fretwise distances, total fretwise distance between each chord and the average of the six surrounding notes, a penalty if this distance is particularly large, total maximum fretwise distance within chords, and total largest fretwise distance between any two notes of sequential chords.

How the Hand is Held. The remaining four factors reward tablatures that allow the hand to remain in the same position as often as possible, allowing each finger to be lifted or pressed exactly once. We look for hand positions that cover the tablature to the maximum extent. For each position, there is a variable that holds the number of notes for which that position can be held and a variable that carries a penalty associated with the difficulty with holding the position.

3.2 Cognitive Factors

There are two other factors that are cognitive in nature, and assess the difficulty that may arise from the violation of certain accepted conventions of guitar performance unrelated to physical difficulty. Guitar players will be startled by tablatures which, even if physically less difficult, are not intuitive. The first of these conventions is that higher notes should be played on higher strings. In line with this convention, a tablature incurs a penalty if a note that is higher than the preceding note is played on a lower string, or a lower note on a higher string. A second convention gives a slight penalty to notes placed higher on the fretboard. This establishes a preference for positions lower on the fretboard in cases where two candidate fretboard positions are of similar difficulty.

4 Improving fitness assessments with the Fitness Function Meta GA (FFMGA)

We have a total of fourteen difficulty factors, and each has an associated weight that determines its contribution to tablature fitness. In previous work we tuned these weights manually, hoping to find a set of weights that would best capture the actual difficulty of tablature. We seek to improve upon our efforts with a Meta GA. The chromosome for a DGA in FFMGA is fourteen continuous valued variables corresponding to the weights of the difficulty factors. Each variable is given a range of possible values that we consider to be reasonable, either 0-30 or 0-3 depending on which order of magnitude we deem most appropriate for that variable. For assessing fitness we have taken from published tablature books and the web a selection of excerpts from 30 different pieces of music comprising a total of 751 notes and their corresponding fretboard positions. We selected the excerpts in a random fashion, though preference was given to excerpts that moved around the fretboard, thereby requiring a more intelligent approach to tablature creation. The fitness of a fitness function chromosome is

the percentage of generated fretboard positions consistent with those assigned by human experts. The implementation of FFMGA is nearly identical to that of the PMGA, and was run for 20000 replacements.

5   LHFNet: a Neural Network for Left Handed Fingering Notation

Tablature sometimes includes information on which specific fingers to use at each fretboard position. This is known as Left Handed Fingering (LHF) notation, and further relieves the performer from having to decide on appropriate performance technique. We have employed a Neural Network to handle this task. We chose to use a Neural Network because of the vast number of inputs that are potentially viable. There is, for each learning pattern, a set of 59 variables that we consider and there is no obvious method for determining which are relevant and how they should be weighted. By using these variables as inputs to a neural network we free ourselves from having to make such determinations. The model we have adopted consults a neural network at every fretboard position in a tablature to determine the appropriate finger to be used. The outputs are values between .1 and .9 for each of the four fingers. We then assign whichever finger has the highest value as the finger for that fretboard position.

5.1   Training Data Acquisition

We trained the network with tablature from the web. Tablature was taken from the popular classical guitar tablature repository at www.classtab.org [16]. We chose thirty pieces of music that included LHF notation. The pieces are all classical, including works from Bach, Villa-Lobos, Tarrega, Sor, and others. In total we generated 6800 training patterns, one for each note, from the data.

5.2   Network Architecure and Inputs

For exploratory purposes, we used the neural network package NeuroShell to get an idea of what network architecture was most suitable for this problem [15]. We tested multilayer, Ward, and simple back-propagation architectures with several combinations of activation functions and layer sizes. The standard three-layer back-propagation architecture with sigmoid activation functions was found to fit the data the best. We will focus our search on networks of this type and use genetic search to determine hidden layer size, learning rate, momentum, and inputs. The neural networks are built with source code from the GAIL package developed by Brian A. Smith [8].

5.3   Optimizing LHFNet with the Neural Network GA (NNGA)

Network Inputs. For each note we have extracted a set of 59 possible network inputs. The first two are the fret and the string of the current note. The next

twenty are comprised of attributes of the most recent note on which each finger was used. These are the fret, string, notes since, chords since, and a binary variable indicating if the note is in the same chord as the current note. There are then four binary variables indicating whether or not each finger is "free", which is dependent on how many notes have elapsed since the finger was last used, and if the string on which the finger was last used has been depressed by a different finger in the intervening time. The next 15 variables are the fret, string and finger of the previous five notes. The next 10 are the fret and string of the next five notes. The remaining eight variables detail the chord that the current note occupies. These are number of notes, number of notes on open strings, the highest string that is open, the number of notes higher than the lowest fret in the chord, the highest played string, the lowest played string, the highest played fret and the lowest played fret.

We extracted all of these features from the data because all are potentially valuable, but we do not know a priori which ones are important. We use NNGA to pare down the inputs for us.

Chromosome and Fitness. The chromosome for one of our networks is a bit string of length 59, two continuous valued variables for learning rate and momentum, and an integer for the number of hidden nodes. Each bit in the bit string determines whether or not an input is included. We set the range for learning rate at .05-.25, momentum at 0-.2 and hidden layer size at 8-17. These ranges were chosen based on what we have observed in exploratory testing with NeuroShell.

Network fitness is determined by accuracy on a production set, which comprises a randomly selected 15% of the 6800 patterns. This is also the size of the test set, and the remaining 70% are used for training. Training has concluded when no improvement in mean squared error has been found in 250000 events.

The GA. NNGA is similar to the Genitor I style used in PMGA and FFMGA. We set population size at 100, mutation rate at 7%, and use rank-based selection with uniform crossover. The GA was run for 30000 replacements.

6 Results

6.1 Optimized DGA vs. DGA vs. Simple GA

We are interested in comparing the effectiveness of the simple GA, the hand-tuned DGA, and the DGA found with PMGA. We ran the three GAs on each piece of music ten times for 20,000 replacements, which is sufficient time for convergence, and recorded how many times each algorithm found the baseline solution. Each GA was run with the parameter settings found to work best for that GA, either empirically or through testing, and all had the same population size.

Table 1. Performance of 3 GAs on 11 musical excerpts. Number of times in 10 runs the baseline solution was found.

	1	2	3	4	5	6	7	8	9	10	11	*Total*
Optimized DGA (TabGA)	10	10	10	10	10	10	7	6	1	0	8	82
Hand Tuned DGA	10	9	10	10	10	10	7	2	1	1	7	77
Simple GA	10	7	7	10	4	3	2	0	0	0	1	44

The DGA parameter set found by PMGA seems to afford us only a marginal benefit, if any. The optimized DGA found the baseline solution in 74.7% of its runs, compared to 70% for the hand tuned DGA. However, there is a noticeable benefit to the DGA over the simple GA, which found the baseline solution only 40% of the time. When we consider only the "difficult" pieces, namely excerpts 5-11, the difference is even more remarkable. On these excerpts the optimized and hand tuned DGAs found the baseline solution in 60% and 56% of cases respectively, while the Simple GA found the solution in only 14.3% of the cases. As may have been expectd, no improvement upon the baseline solutions was made.

6.2   Consistency of TabGA and LHFNet with human experts

The accuracy achieved by TabGA on our test set is 91.1%. That is, 91.1% of the fretboard positions in the generated tablature were consistent with the human-created tablature. This is a marked improvement over the DGA using our previous fitness function, which has an accuracy of 73% on the same set of data. In addition, deviations from published tablature did not result in significantly more difficult tablature. For LHF notation, the accuracy achieved by LHFNet on the independent test set was 80.6%.

The University of Torino reports accuracies of 98% and 90% respectively, but these numbers were achieved on a very different set of data. The best tablatures that we were able to find on the web for the same pieces of music on which their system was judged had the characteristic that nearly every note was placed at the lowest possible position. Tablatures of this nature are not particularly difficult to create, and our system achieved an accuracy of 98.9% on data from the same pieces of music. We could not test our neural network on this data, however, since left handed fingering notation for these pieces was not available to us. For most of the thirty excerpts on which our algorithm was run, tablature creation is much more difficult. For this reason, we contend that the versatility and validity of our approach can be justly asserted.

7 Further Directions

This system has been used in the development of a framework for creating arrangements of pieces of music never before written for guitar, with compelling results. It acts as part of an evaluation function for a guided heuristic search that distinguishes arrangements on the basis of both playability, as defined by TabGA, and musical merit[10].

References

1. Gordon, V.S., Whitley, D.: Serial and Parallel Genetic Algorithms as Function Optimizers. 5th International Conference on Genetic Algorithms (1993) 177–183
2. Heijink, H., Meulenbroek, R.G.J.: On the complexity of classical guitar playing: functional adaptations to task constraints. Journal of Motor Behaviour **34(4)** (2002) 339–351
3. Miura, M., Hirota, I., Hama, N., Yanigida, M.: Constructing a System for Finger-Position Determination and Tablature Generation for Playing Melodies on Guitars. Systems and Computers in Japan **35(6)** (2004) 755–763
4. Radicioni, D., Lombardo, V.: Guitar Fingering for Music Performance. Proceedings of the Internation Computer Music Conference. Barcelona, Spain. (2005)
5. Radicioni, D., Anselma, L., Lombardo, V.: A segmentation-based prototype to compute string instruments fingering. Proceedings of the Conference on Interdisciplinary Musicology. Graz, Austria. (2004)
6. Radisavljevic, A., Driessen, P.: Path Difference Learning for Guitar Fingering Problem. Proceedings of the International Computer Music Conference. Miami, USA. (2004)
7. Sayegh, S.: Fingering for String Instruments with the Optimum Path Paradigm. Computer Music Journal **13(6)** (1989) 76–84
8. Smith, B.: GAIL: Georgia Artificial Intelligence Library Neural Network Package. University of Georgia. (2004)
9. Tuohy, D., Potter, W.D.: A Genetic Algorithm for the Automatic Generation of Playable Guitar Tablature. Proceedings of the International Computer Music Conference. Barcelona, Spain. (2005)
10. Tuohy, D., Potter, W.D.: GA-based Music Arranging for Guitar. Submitted to IEEE Congress on Evolutionary Computation (2006).
11. Wang, J., Tsai-Yen, L.: Generating Guitar Scores from a MIDI Source. International Symposium on Multimedia Information Processing. Taipei, Taiwan. (1997)
12. Whitley, D.: Genetic Algorithms and Neural Networks. Genetic Algorithms in Engineering and Computer Science (1995) 1–15
13. Whitley, D., Kauth, J.: GENITOR: A Different Genetic Algorithm. Proceedings of the Rocky Mountain Conference on Artificial Intelligence.
14. Whitley, D., Starkweather, T.: GENITOR II: a distributed genetic algorithm. J. Expt. Theor. Artif. Intel. **2** (1990) 189–213
15. NeuroShell 2 Neural Network Kit by Ward Systems Group. http://www.wardsystems.com.
16. Classical Guitar Tablature. http://www.classtab.org
17. A-Z Guitar Tabs. http://www.guitaretab.com